

Bristle Blocks:

A Silicon Compiler

Dave Johannsen

Caltech

Abstract

Standard LSI Design Automation systems are database management systems that aid the circuit designer by organizing the collection of submodules that comprise a chip. This type of file system usually does not aid in the actual computation of silicon layout, and can hinder a designer with program constraints that have little or nothing to do with silicon constraints. The Bristle Block system is an attempt to create a silicon compiler that will perform the majority of the implementation computation while placing a minimum set of constraints on the designer. The goal of the Bristle Block system is to produce an entire LSI mask set from a single page, high level description of the integrated circuit.

Introduction

After designing a few reasonable sized LSI integrated circuits, it becomes painfully obvious that the current design tools aid little in the every increasing task of automating chip design. Most systems are little more than fancy filing cabinets that help organize the data associated with various elements composing a chip. Even with these advanced systems, 90% of the design is completed in 10% of the time, while the remaining 10% of the design requires the remaining 90% of the time. Too many design decisions are "frozen in concrete" early in the game, before the effects of those decisions become apparent.

What if a person were able to sit down and design a complete chip in a single afternoon? The user would grab a few building blocks, snap them together, and experiment with many radically differing designs before deciding upon the actual implementation. What if that person were given complete mask layouts and simulations for each of his or her experimental configurations with almost no effort? This is the environment that the Bristle Blocks system attempts to provide.

Design Goals

The goal of the Bristle Block system is to allow the user to design LSI integrated circuits with as little concern for the mechanics involved as possible. The system must be capable of designing fairly complicated chips, and the resulting designs should have a high degree of optimization in the layout. The results should be competitive with "hand layout" chips.

Structured Design. The Bristle Block system is intimately dependant on the structured design methodology, as presented in Mead and Conway [1]. This system encourages the design of regular computing structures. These regular computing structures tend not to limit the power or usefulness of the resulting machines, but rather to enhance the performance and generality of the hardware.

Hierarchical Design. The chips designed by Bristle Blocks are hierarchical in nature. This hierarchy imposes a locality on the various sections of the chip that can be exploited when performing design rule verifications and electrical simulations of the chip.

Various Representations. The Bristle Block system provides various representations for the integrated circuit. The representations span the entire range from the physical to the conceptual aspects of the chip. Bristle Blocks is designed to handle the following seven representations:

LAYOUT. The "lowest" level of representation is the Layout level, which produces the actual chip masks.

STICKS. The Sticks level produces a *stick diagram* of the chip, which has the same topology as the layout, but with all of the features reduced to single-width lines. The resulting diagram is much easier to comprehend than the full layout diagram.

TRANSISTORS. The Transistor level produces a transistor diagram for the chip or subsection of the chip.

LOGIC. The Logic level of the chip can produce a logic diagram of the chip in the TTL style.

TEXT. The Text level prints a hierarchical description of the chip that can be used as a "user's manual" for the completed chip.

SIMULATION. The Simulation level can be used to logically simulate the chip, so that software can be written for the chip to explore the feasibility of the design.

BLOCK. The Block level draws a block diagram of the chip, showing the arrangement of the buses and core elements.

Every fundamental element in the Bristle Block system has the capability of containing each of these seven representations for itself. When the system is physically connecting the elements, the logical and textual links are also generated. Other representations may be added to the system at any time.

Bristles and Blocks

The fundamental unit in the Bristle Block system is the cell, which may contain geometrical primitives and references to other cells. These cells to the LSI designer can be equated to the programmer's subroutines: each contain a few primitive operations and references to lower levels in the hierarchy. The geometric primitives are instances of lines, boxes, and polygons, each with an associated mask layer.

Bristle Blocks also has a structure for specifying the location and flavor of connection points between cells. These connection points are like bristles along the edges of the cells, and it is upon these bristles that the Bristle Block system builds most of the computable structures. Connection points help keep local data local and global data global, while delaying the binding of many design constraints. For instance, a cell that requires an input from a pad would contain a connection point stating where in the cell the pad should connect and what type of pad is needed. When the chip is compiled, the appropriate pad is automatically placed on the chip and a wire is routed between the pad and the cell. Thus, the local data, which declare where in the cell the pad connects and the type of pad, is kept local to the cell, while the global data, which specify where the pad is located and how the wire is routed, is kept global to the cell. Also, none of the data need be supplied by the user at compile time.

The data necessary to specify the various representations for the cells and connection points may be stored in disk files and read in as needed, to allow for the use of common cell libraries and sharing of data. Any low level cell that the user may need for his chip must be entered into either the system or a library before the chip can be compiled. Associated with each cell is the information necessary to extract the cell's definition from a file, if such a file exists.

The low level cells in a library are defined by entering the actual layout of each cell representation in a standard cell design language. The low level cell design task was not given to the compiler, but left to the user, for three reasons:

- 1) The design of low level cells does not take much time. Each cell usually has a small area, few transistors, and is reasonably well defined.
- 2) Very few mistakes are made in the design of low level cells. Experience has shown that virtually none of the fatal chip errors occur in the low level cells, but that chips fail because of faulty "glue": the high level cells holding the chip together.

3) Human ingenuity pays off well in the low level cell design. There has yet to be written a computer program that approaches human ingenuity in cell design, which is most noticable in the low level cells.

Chip Format

As a starting point, one style of chip design was selected to explore the possibilities of a Bristle Block system. Although it was initially felt that this style would be rigid and special purpose, the generality of the style became apparent as the potentials of Bristle Blocks were discovered. The style of chip design is stated in the description of three formats of the chips: the physical, logical, and temporal formats.

Physical Format. The physical format of chips built by the current Bristle Block system is shown in figure 1. A chip consists of a central *core*, which is controlled by an *instruction decoder*, both of which are surrounded by pads. The core is composed of a series of data processing elements, such as memories, shifters, and arithmetic-logic units. Based upon the requirements of the core, the instruction decoder and pads are automatically generated and placed in the final chip.

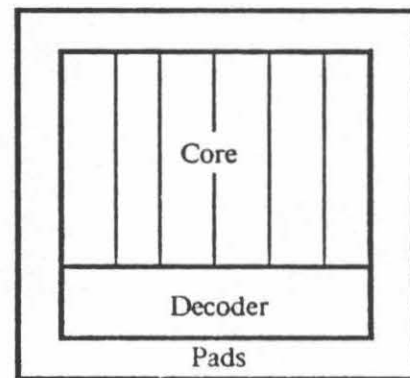


Figure 1. Physical Chip Format

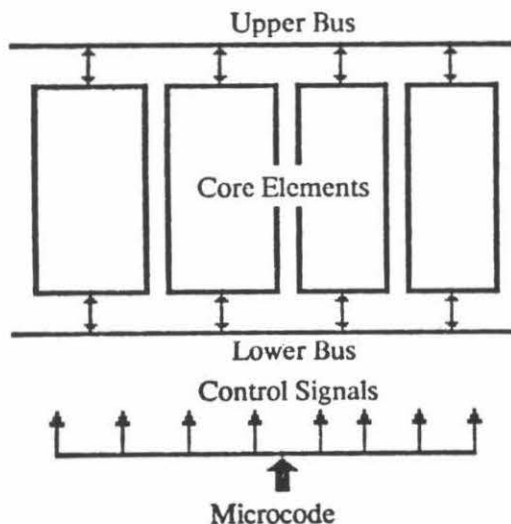


Figure 2. Logical Chip Format

Logical Format. The logical format of the chips is shown in figure 2. Each of the core elements can communicate with either of two buses that run through the elements. These buses may run the length of the chip, or they may stop anywhere along the chip with new buses servicing the remainder of the chip. The order of placement of elements along the core is irrelevant to the system, with the exception that at most two buses may run through any element. The microcode words which control the operation of the chip enter the decoder twice during each clock cycle. The appropriate control functions are derived from these microcode words and latched by control signal buffers. These buffers then drive the control lines of each core element.

Temporal Format. The chip is driven by a two phase, non-overlapping clock. One phase, referred to as $\phi 1$, controls the transfer of data between elements via the buses. The alternate clock phase, referred to as $\phi 2$, controls the operation of the data processing elements. During $\phi 2$, the buses are precharged to a high state. Any processing element that is activated during $\phi 1$ may pull the bus lines low. Any element may read the data from the bus by the end of $\phi 1$. While the buses are transferring data, the data processing elements may be precharged. An example of a precharging processing element is an adder, where the carry chain is precharged to a high state. During the following $\phi 2$, the various points along the carry chain may be pulled to a low state. Instructions enter the control buffers through the decoder logic on the clock phase preceding the phase when the instruction is to be executed.

Operation

Bristle Blocks is a three pass compiler, consisting of a core pass, a control pass, and a pad pass. The core pass takes both the user's input and low level cell definitions to construct the core of the machine. The control pass adds the instruction decoder to the core by generating a decoder which fills the constraints posed by *control connection points* in the core. The pad pass adds the pads to the perimeter of the chip and routes wires between the pads and the corresponding connection points in the core and decoder.

User Input

The input to the compiler consists of three sections. The first section states the microcode instruction width and describes the decomposition of the microcode word into various fields, such as the "Register Select Field" or the "ALU Operation Field." The second section states the data word width for the chip and lists the buses that run through the core of the chip. The final section lists the elements of the chip's core, and provides any parameter values that the elements require.

Pass 1: Core Layout

The core pass is driven by the user's input. The input contains the list of elements and associated parameters. After all of the elements vote on the values of global parameters, each element is executed in turn, resulting in a hierarchy of cells which implement the core of the chip.

To allow any two elements to plug together, cells must interface either by maintaining a common pitch (width) or by wire routing. To save the space and costly routing needed if cell widths vary, a design constraint states that all cells must be of equal width. To have a uniform width for all cells, every cell must be *designed* as wide as the widest cell. The width of the widest cell is not known until *after* all of the cells are designed! And as future cells are designed, they must either be forced to have the same width as current cells, or else all of the cells must be redesigned to accommodate the wider cells.

By introducing stretchable cells, this problem can be avoided. Each of the cells are designed with places to stretch. As the core is being scanned, each element reports the width of its cells, so that when the end of the core list is reached, the widest cell width is known. As the elements produce their cells, each cell is stretched (a painless operation) to fit all other cells. The cells can also be stretched to allow the power lines to expand as power demands increase.

Another restriction arising from the arbitrary element ordering is that each of the cells must meet certain interface requirements. Proper interface standards eliminate intercell problems such as shorting out a neighbor's transistor or power supply line. By agreeing on a standard interface to begin with, any cell can be guaranteed to mesh properly with adjacent cells before the neighboring cells are specified. Boundary conditions like these allow design rule checking to be performed on individual cells as the cells are designed, rather than on fully instantiated artwork.

Along with meshing of cells, the busing requirements between cells must also be met. Buses may need to break or stop, and bus precharge circuits must be added for each bus. Details like these need not be specified by the user, but are added by the compiler.

Pass 2: Control Design

Given the results of the core pass, the control design and layout proceeds. First, control buffers to drive the control lines are inserted along the edge of the core. The timing is also added to the control signals by the buffers. Then, an text array is constructed which specifies the decode functions needed for each buffer. A two-tape Turing machine operates on one "tape", which contains the text array, and writes the second "tape", producing compiled silicon code. When it has finished operating on the array, the Turing machine will have generated and optimized the instruction decoder, and created pad connections for the inputs to the decoder.

Pass 3: Pad Layout

The pad layout pass of the silicon compiler begins by collecting all of the connection points which need to be connected to pads. These connection points are sorting in clockwise order, and pads are allocated in the same order. The pads and connection points are examined by a Roto-Router, which rotates the pads around the perimeter of the chip in an attempt to minimize the length of wire between pads and connection points. The Roto-Router spaces the pads evenly around the chip to avoid generating pad layouts that would be difficult to bond. The third pass concludes by adding wires between the pads and the connection points.

Conditional Assembly

Bristle Blocks can allow for conditional assembly of silicon. For example, when designing prototype chips, the internal state of a state machine may need to be routed to pads, but when production chips are produced, the area of the pad and wires may need to be reclaimed. The user may declare

a global boolean variable `PROTOTYPE`, which, if `TRUE`, will add the connection points for the pads, but if `FALSE` will not. At any time prior to actually compiling the chip, the user may decide whether this is a prototype chip or not, and properly modify the layout.

Since Bristle Blocks provides a high level programming language to the user, cells can be smart, and perform calculations as they are being added to the core. As an example, there may be several possible cell layouts which implement the desired function. After the cell width has been determined, the possible layouts which fit within the specified width can be judged to find the cell with minimum resulting area. The minimum area cell is then used in the actual chip, which optimizes the length of the chip.

Implementation

Bristle Blocks is implemented in ICL, a high level graphics, integrated circuit design, and general purpose programming language created by Ron Ayres [2]. ICL allows the user to generate new data structures at any time. Functions and coercion can also be added to manipulate the data structures. The coercions are invoked automatically by ICL when the data it receives are not of the type it expects. Also, ICL provides for polymorphic functions, where the operation performed by the function is dependent upon the data supplied to the function. ICL is an extensible language, so that any user defined functions, datatypes, and coercion become a part of the current version of ICL. This also means that the Bristle Block system provides the user with all of the basic ICL operations, and that Bristle Blocks is an extensible system..

The only disadvantage with using ICL is that the current version is an implementation for a PDP10/PDP20, which has a small (18 bit) address space.

Conclusion

In the 2½ man-months that have gone into developing the first Bristle Blocks system, approximately 80% of the system has been implemented. Given a high level description of the chip and definitions for core elements, the system produces a complete layout, sticks diagram, transistor diagram, logic diagram, and block diagram. There are a few special cases in the circuit topology that have still to be considered, but lack of time is the only reason they have not been implemented. Hooks for the circuit simulator and text producing code have been included in the system, but with the constraint that all code must remain in core, these features must wait.

The chips produced by the system are fairly well optimized, having $\pm 10\%$ of the area of a chip produced by hand using the structured design methodology. The compiler takes approximately 4 minutes to generate a small chip, in all five of the current representations. The time needed to generate a fairly large chip should be in the neighborhood of 10-15 minutes.

At present, the Bristle Blocks system is tailored to produce chips of a particular architecture. As the domain of silicon compilers is examined through the use of the current Bristle Blocks, a more general chip architecture will be found that encompasses the current chip architecture. A more generalized Bristle Blocks will be developed to compile the generalized chip architecture. The generalization of the current Bristle Block system may continue through a number of generations. At some point, however, a totally new Bristle Blocks will be generated to handle a completely different style of chip. Thus, there will emerge several classes of Bristle Block systems, each comprising a separate class of chip architectures, rather than one Super Bristle Block system which attempts to produce *all* chips.

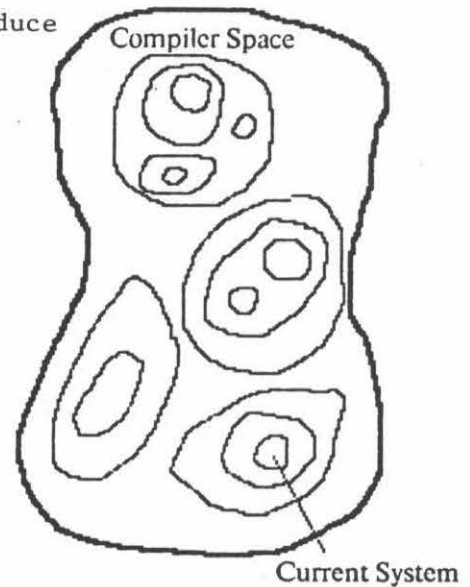


Figure 3. Hierarchy of Systems

References

- [1] Mead, Carver A., and Conway, Lynn "An Introduction to VLSI Systems," limited printing, 1978
- [2] Ayres, Ronald, "ICL Reference Manual," SSP File #1364, CalTech, 1978